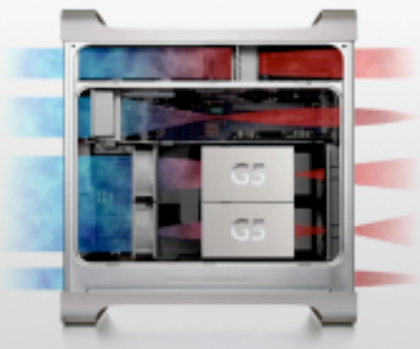
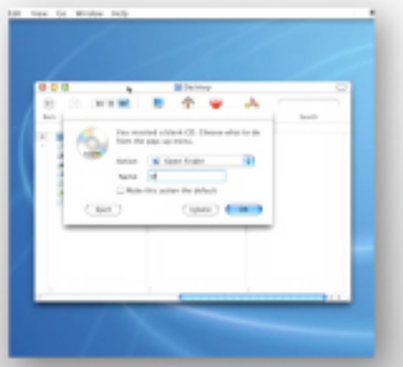
HID and Force Feedback

Session 509





HID and Force Feedback

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

Introduction

HID Manager

Rob Yepez
I/O Kit Team
ryepez@apple.com

Introduction

- What is HID
- HID Architecture on Mac OS X
- Improvements since 10.2
- Techniques for using the HID Manager

What You Will Learn

- How HID fits into the system
- How to effectively use the HID Manager

What Is HID

- HID = Human Interface Device
- Developed by the USB-IF
- Devices humans use to interact with a computer

Device Examples

- Keyboards and pointing devices
- Front-panel controls
- Game or simulation devices
- Devices that may not require human interaction

Terminology—HID Reports

- Raw data sent to or received from the device
- Polled or interrupt driven
- Types of reports
  - Input
  - Output
  - Feature

Terminology—Elements

- Used to represent parsed items from a report
- Behavior defined by usage page and usage
- Types of elements
  - Data
  - Input
  - Output
  - Feature
- Collection

Terminology—Collections

- Represents relationship between two or more data elements
- Types of collections
  - Physical
  - Application
  - Logical
  - Report
  - Named Array

Example Mouse Descriptor

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Report Count (3),
        Report Size (1),
        Usage Page (Buttons),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Report Size (8),
        Report Count (2),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

Example Mouse Descriptor

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Report Count (3),
        Report Size (1),
        Usage Page (Buttons),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Report Size (8),
        Report Count (2),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

Example Mouse Descriptor

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Report Count (3),
        Report Size (1),
        Usage Page (Buttons),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Report Size (8),
        Report Count (2),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

Example Mouse Descriptor

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Report Count (3),
        Report Size (1),
        Usage Page (Buttons),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Report Size (8),
        Report Count (2),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

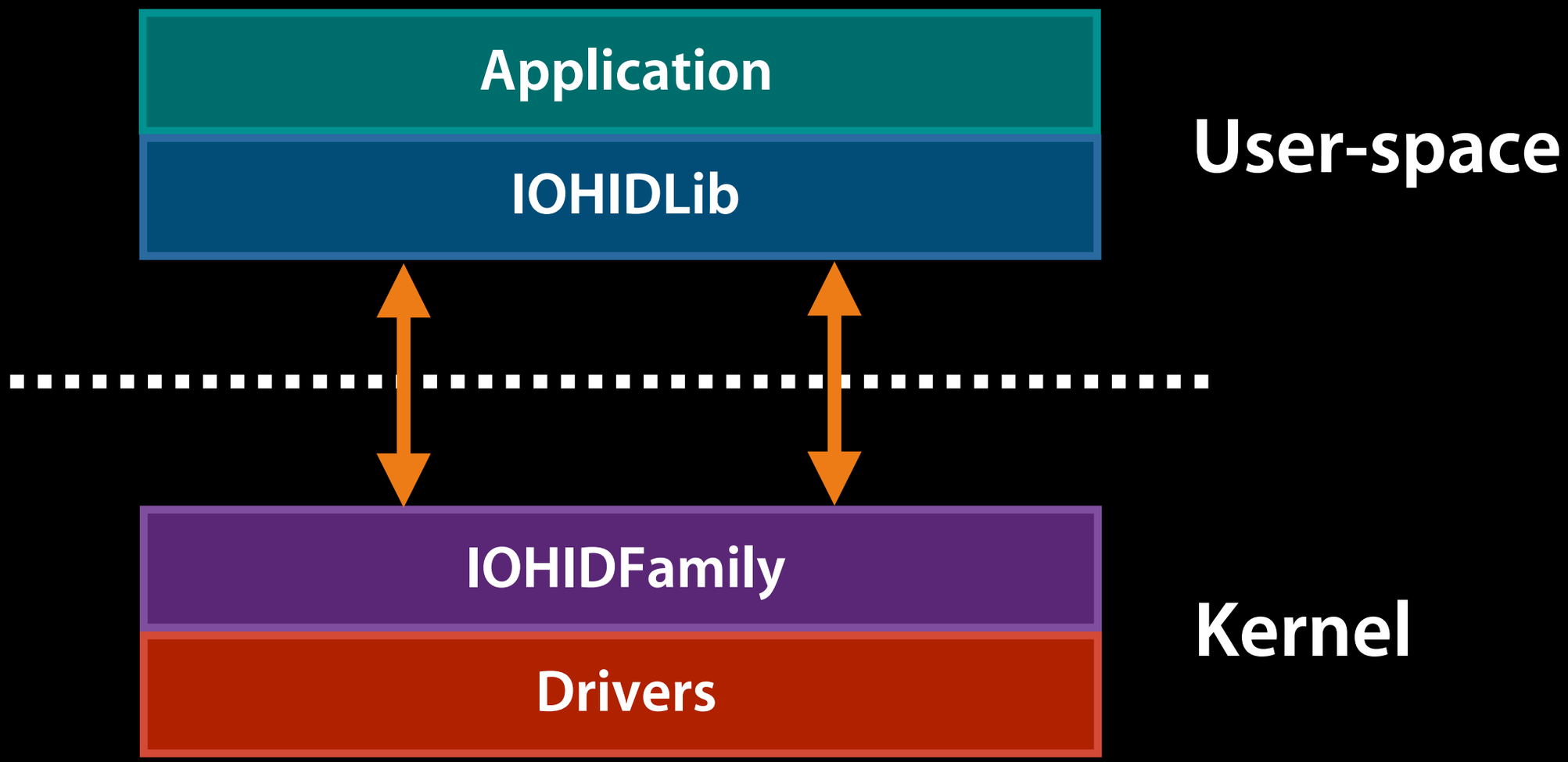
Example Mouse Descriptor

```
Usage Page (Generic Desktop),
Usage (Mouse),
Collection (Application),
    Usage (Pointer),
    Collection (Physical),
        Report Count (3),
        Report Size (1),
        Usage Page (Buttons),
        Usage Minimum (1),
        Usage Maximum (3),
        Logical Minimum (0),
        Logical Maximum (1),
        Input (Data, Variable, Absolute),
        Report Count (1),
        Report Size (5),
        Input (Constant),
        Report Size (8),
        Report Count (2),
        Usage Page (Generic Desktop),
        Usage (X),
        Usage (Y),
        Logical Minimum (-127),
        Logical Maximum (127),
        Input (Data, Variable, Relative),
    End Collection,
End Collection
```

What Is HID Manager

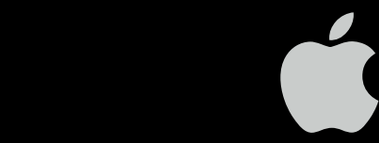
- Interface for receiving raw HID events from a device
- It consists of 3 layers:
  - HID Drivers
    - `IOUSBHIDDriver`
    - `IOBluetoothHIDDriver`
  - Kernel architecture and libraries
    - `IOHIDFamily`
  - User space API
    - `IOHIDLib`

HID Manager Architecture



HID Driver

- Based on abstract class `IOHIDDevice`
- Transport not restricted to USB—opportunity to provide HID support for:
  - Serial
  - FireWire
- Normalized on USB HID specification



IOHIDFamily

- Parses HID reports
- Interacts directly with the IOHIDLib
- Maintains shared memory
- Publishes device properties in the IORegistry
  - Keys and values defined in IOHIDKeys.h
    - Device-wide
      - Constants start with `kIOHID`
    - Element-wide
      - Constants start with `kIOHIDElement`

IOHIDDevice

- Example device IORegistry entry

```
{
   "VendorID"              = 1452,
   "LocationID"            = 1048576,
   "Transport"             = "USB",
   "Product"               = "M4848",
   "VersionNumber"         = 1280,
   "PrimaryUsage"          = 2,
   "PrimaryUsagePage"      = 1,
   "Manufacturer"          = "Logitech",
   "ProductID"             = 769,
   "Elements"              = (Array of Elements)
}
```

IOHIDDevice

- Example device IORegistry entry

```
{
   "VendorID"              = 1452,
   "LocationID"            = 1048576,
   "Transport"             = "USB",
   "Product"               = "M4848",
   "VersionNumber"         = 1280,
   "PrimaryUsage"          = 2,
   "PrimaryUsagePage"      = 1,
   "Manufacturer"          = "Logitech",
   "ProductID"             = 769,
   "Elements"              = (Array of Elements)
}
```

IOHIDDevice

- Example collection element IORegistry entry

```
{
   "ElementCookie"              = 1,
   "Usage"                      = 2,
   "UsagePage"                  = 1,
   "Type"                       = 513,
   "CollectionType"             = 1,
   "Elements"                   = (Array of elements)
}
```

IOHIDDevice

- Example data element IORegistry entry

```
{
    "ElementCookie"                 = 3,
    "UsagePage"                     = 9,
    "Usage"                         = 1,
    "Type"                          = 2,
    "Size"                          = 1,
    "Max"                           = 1,
    "Min"                           = 0,
    "ScaledMax"                     = 1,
    "ScaledMin"                     = 0,
    "Unit"                          = 0,
    "UnitExponent"                  = 0,
    "IsArray"                       = No,
    "IsRelative"                    = No,
    "IsNonLinear"                   = No,
    "IsWrapping"                    = No,
    "HasPreferredState"             = Yes,
    "HasNullState"                  = No,
    "ValueLocation"                 = 192,
}
```

IOHIDLib

- CFPlugIn Architecture
- API currently broken into 3 interfaces
  - `IOHIDDeviceInterface`
  - `IOHIDQueueInterface`
  - `IOHIDOutputTransactionInterface`
- Each interface defined by a UUID

IOHIDDeviceInterface

- Establish connection to device
- Read and post element values
- Standard APIs
  - `getElementValue`
  - `queryElementValue`
  - `setElementValue`
- Advanced APIs
  - `getReport`
  - `setReport`

IOHIDQueueInterface

- Useful when application needs to respond to every HID event
- Polled or setup to receive asynchronous notifications
- APIs include
  - `getNextEvent`
  - `setEventCallout`

IOHIDOutputTransactionInterface

- Consolidates HID reports used for setting multiple element values
- Replacement for setReport
- APIs include
  - `setElementValue`
  - `setElementDefault`
  - `commit`

HID Manager Improvements

- Ability to seize HID events from a device
- New IOHIDDeviceInterface API
  - New in 10.2.3
    - `getReport`
    - `setReport`
  - New in Panther
    - `copyMatchingElements`
    - `setInterruptReportHandlerCallback`

Techniques—Getting Started

- Find matching devices

```
// Get a Mach port to initiate communication
IOMasterPort (bootstrap_port, &mastPort);

hidMatchDictRef = IOServiceMatching (kIOHIDDeviceKey);

// Narrow search by adding items to matching dictionary
usagePage        = kHIDPage_GenericDesktop;
usage            = kHIDUsage_GD_Mouse;
usagePageNumRef  = CFNumberCreate (kCFAllocatorDefault, kCFNumberIntType, &usagePage);
usageNumRef      = CFNumberCreate (kCFAllocatorDefault, kCFNumberIntType, &usage);

CFDictionarySetValue (hidMatchDictRef, CFSTR (kIOHIDPrimaryUsagePageKey), usagePageNumRef);
CFDictionarySetValue (hidMatchDictRef, CFSTR (kIOHIDPrimaryUsageKey), usageNumRef);

IOServiceGetMatchingServices (masterPort, hidMatchDictRef, &hidObjectIterator);
```

Techniques—Getting Started

- Iterate through the matching devices

```
io_object_t  hidDevice = NULL;

while ( hidDevice = IOIteratorNext (hidObjectIterator) )
{
     //See next slide for creating device interface
     ..
     ..
     ..

     IOObjectRelease(hidDevice);
}
```

Techniques—Getting Started

- Create an interface to the device

```
IOHIDDeviceInterface122 **   hidDeviceInterface = NULL;;
IOCFPlugInInterface **       plugInInterface = NULL;
SInt32                       score = 0;

IOCreatePlugInInterfaceForService(hidDevice,
                            kIOHIDDeviceUserClientTypeID,
                            kIOCFPlugInInterfaceID,
                            &plugInInterface,
                            &score);

(*plugInInterface)->QueryInterface(plugInInterface,
                            CFUUIDGetUUIDBytes(kIOHIDDeviceInterfaceID122),
                            &hidDeviceInterface);

//See next slide for using the interface
..
..
..
```

Techniques—IOHIDDeviceInterface

- Opening the device

```
// Open the device with no options
(*hidDeviceInterface)->open (hidDeviceInterface, 0);
```

- Seizing the device

```
// Open the device with the seize option
(*hidDeviceInterface)->open (hidDeviceInterface, kIOHIDOptionsTypeSeizeDevice);
```

Techniques—IOHIDDeviceInterface

- Finding elements

```
CFMutableDictionaryRef      elementMatchDictRef = NULL;
CFArrayRef                  matchedElementArrayRef = NULL;

elementMatchDictRef = CFDictionaryCreateMutable(kCFAllocatorDefault, 0,
                            &kCFTypeDictionaryKeyCallBacks,
                            &kCFTypeDictionaryValueCallBacks);

// Look for all the button elements
usagePage        = kHIDPage_Button;
usagePageNumRef  = CFNumberCreate (kCFAllocatorDefault, kCFNumberIntType, &usagePage);

CFDictionarySetValue (elementMatchDictRef,
                            CFSTR (kIOHIDElementUsagePageKey),
                            usagePageNumRef);

(*hidDeviceInterface)->copyMatchingElements (hidDeviceInterface,
                                             elementMatchDictRef,
                                             &matchedElementArrayRef);

// See next slide for cookie storage
..
..
..
```

Techniques—IOHIDDeviceInterface

- Store element cookies

```
int                     buttonCount = CFArrayGetCount (matchedElementArrayRef);
IOHIDElementCookie      buttonCookies[buttonCount];
CFDictionaryRef         element;
CFNumberRef             number;

for (int i=0; i< buttonCount; i++)
{
     element = CFArrayGetValueAtIndex (matchedElementArrayRef, i);

     number = CFDictionaryGetValue (element, CFSTR(”kIOHIDElementCookieKey”));

     CFNumberGetValue(number, kCFNumberLongType, &buttonCookies[i]);
 }
```

- Get the element value

```
IOHIDEventStruct        event;
Sint32                  value;

// Get the value of the first button
(*hidDeviceInterface)->getElementValue (hidDeviceInterface, buttonCookies[0], &event);

value = event.value;
```

Techniques—IOHIDQueueInterface

- Create queue

```
IOHIDQueueInterface **      hidQueueInterface = NULL;

// Allocate a queue
hidQueueInterface = (*hidDeviceInterface)->allocQueue (hidDeviceInterface);

// Create the queue 8 items deep
(*hidQueueInterface)->create (hidQueueInterface, 0, 8);
```

- Add elements to the queue

```
// Add the button elements to the queue
for (int i=0; i<buttonCount; i++)
{
    (*hidQueueInterface)->addElement (hidQueueInterface, buttonCookies[i], 0);
}
```

Techniques—IOHIDQueueInterface

- Setup queue for asynchronous operation

```
CFRunLoopSourceRef    evSource;

// Create the event source
(*hidQueueInterface)->createAsyncEventSource (hidQueueInterface, &evSource);

// Add the event source to the run loop
CFRunLoopAddSource (CFRunLoopGetCurrent(), evSource, kCFRunLoopDefaultMode);

// Set the queue callback
(*hidQueueInterface)->setEventCallout (hidQueueInterface,
                            MyHIDQueueCallbackFunction,
                            0,
                            hidQueueInterface);
```

Techniques—IOHIDQueueInterface

- Remove items from the queue

```
void MyHIDQueueCallbackFunction(void * target, IOReturn result, void * refcon, void * sender)
{
    AbsoluteTime            zeroTime = {0,0};
    IOReturn                result;
    IOHIDEventStruct        event;
    SInt32                  value;

    do
    {
          result = (*hidQueueInterface)->getNextEvent(hidQueueInterface, &event, zeroTime, 0);

          value = event.value;

          // Do something with the value

    } while (result == kIOReturnSuccess);

}
```

Techniques—IOHIDOutputTransactionInterface

- Create output transaction

```
IOHIDOutputTransactionInterface ** hidOutTransactionInterface = NULL;

// Allocate a queue
hidOutTransactionInterface = (*hidDeviceInterface)->allocOutputTransaction(hidDeviceInterface);

// Create the queue 8 items deep
(*hidOutTransactionInterface)->create (hidOutTransactionInterface, 0, 8);
```

- Add elements to the transaction

```
// Add the led elements to the transaction
for (int i=0; i<ledCount; i++)
{
    (*hidOutTransactionInterface)->addElement (hidOutTransactionInterface, ledCookies[i], 0);
}
```

Techniques—IOHIDOutputTransactionInterface

- Set the element defaults

```
IOHIDEventStruct      event;

// Zero out the event
bzero(&event, sizeof(IOHIDEventStruct);

// set the default value for all elements
for (int i=0; i<ledCount; i++)
{
    (*hidOutTransactionInterface)->setElementDefault (hidOutTransactionInterface,
                                    ledCookies[i], event);
}
```

Techniques—IOHIDOutputTransactionInterface

- Set the element values

```
// turn on the capslock led
event.value = 1;
(*hidOutTransactionInterface)->setElementValue (hidOutTransactionInterface,
                                ledCookies[kHIDUsage_LED_CapsLock],
                                                event);

// turn on the numlock led
event.value = 1;
(*hidOutTransactionInterface)->setElementValue (hidOutTransactionInterface,
                               ledCookies[kHIDUsage_LED_NumLock],
                                               event);
```

- Commit the transaction

```
// commit the transaction
(*hidOutTransactionInterface)->commit (hidOutTransactionInterface, 0, 0, 0, 0);
```

Summary

- Human Interface Devices
- HID on Mac OS X Architecture
- Techniques for using the HID Manager

Additional References

- Apple Developer documentation
  - http://developer.apple.com/techpubs/macosx/Darwin/IOKit/DeviceInterfaces/HID/HIDDocPage.html
  - http://developer.apple.com/techpubs/macosx/Darwin/General/WritingDeviceDriver/index.html
- USB-IF HID pages
  - http://www.usb.org/developers/hidpage/

Force Feedback

Fernando Urbina
USB Software Team
nano@apple.com

Introduction

- What is Force Feedback
- Force Feedback architecture on Mac OS X
- Force Feedback for Application Developers
- Force Feedback for Hardware Vendors

What You Will Learn

- How Force Feedback fits in the system
- Why we did it that way
- How you can take advantage of Force Feedback
- How you can develop a hardware plug-in

What Is Force Feedback

- A technology that enhances the human/computer interface by engaging your sense of touch
- Not limited to games
- Immersion licenses their TouchSense™ technology to hardware developers

Terminology—Force Feedback Device

- Force Feedback Device
  - USB HID Device (sometimes Physical Interface Device compliant)
  - Capable of creating a force (effect) that user can feel
  - These effects are played through software
    - Programmatically
    - Response to a trigger on the device (button press)

Terminology—Types of Effects

- Four basic types of effects
  - Constant Force
  - Ramp
  - Periodic
  - Condition (Spring, Damper, Inertia)

Examples of Effects

- Racing/flight simulations
  - Road texture (paved, dirt, stone)
- Non-gaming applications
  - Educational/physics related

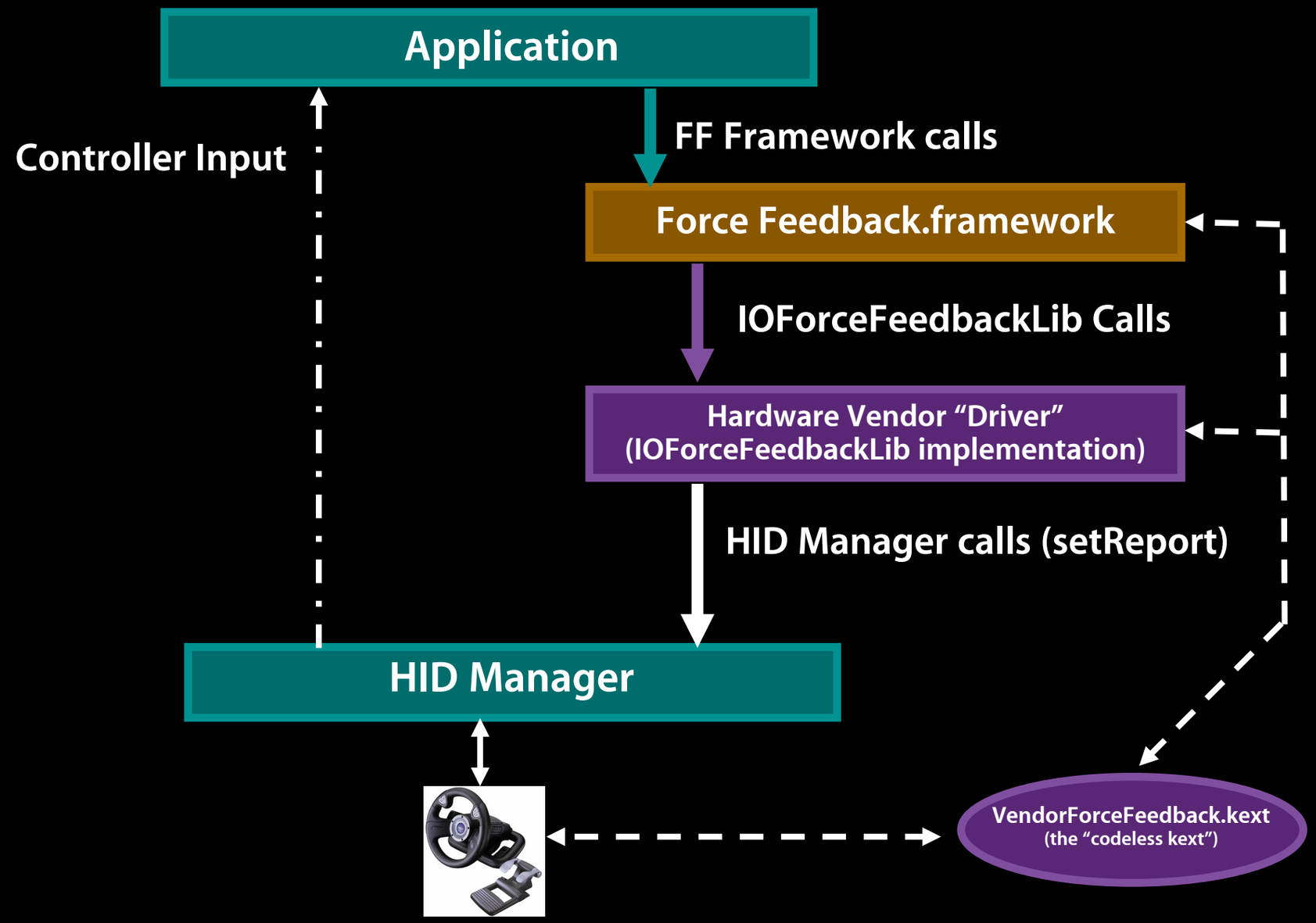
Force Feedback on Mac OS X

- Worked with Immersion to bring Force Feedback support to Mac OS X
- ForceFeedback.framework and hardware plug-ins
- Shipped with version 10.2.3 in December 2002
  - Devices with Immersion TouchSense™ technology
    - Logitech, Immersion, Saitek, and others
  - F1 Championship and Nascar Racing 2002

Force Feedback Goals

- Time to market
- Target developers
  - Porting houses for games
    - Make it easy for them to port the software
  - Hardware manufacturers
    - Allow them to control their devices
- Avoid reinventing the wheel
- Leverage Immersion's expertise

Force Feedback Architecture



ForceFeedback.framework

- Fairly similar to the FF API in other architectures
- API categories
  - System-wide
    - API Name starts with `FF`
  - Device-wide (deals with all effects on a device)
    - API Name starts with `FFDevice`
  - Effect-wide (deals with a single effect)
    - API Name starts with `FFEffect`

System-Wide FF APIs

- Used to deal with a FF device
- `FFIsForceFeedback()`
- `FFCreateDevice()`
- `FFReleaseDevice()`

Device-Wide FF APIs

- Deals with the effects on a single device
- Examples of API
  - `FFDeviceCreateEffect() and FFDeviceReleaseEffect()`
  - `FFDeviceGetForceFeedbackCapabilities()`
  - `FFDeviceSendForceFeedbackCommand()`

Effect-Wide FF APIs

- Used to download, play, stop, modify, or destroy an effect that is on a FF device
- Examples of API
  - `FFEffectDownload()`
  - `FFEffectGetParameters()`
  - `FFEffectStart()`
  - `FFEffectUnload()`

Force Feedback Structures

- The devil is in the structures used by the API
- Very similar to FF structures in other platforms
- HeaderDoc is very complete in the description of the API and the structures/constants

Example of Setting Up FFEffect

```
ffperiodic.dwMagnitude = FF_FFNOMINALMAX;
ffperiodic.lOffset = 0;
ffperiodic.dwPhase = 0;
ffperiodic.dwPeriod = 200000;

ffeffect.dwSize = sizeof( ffeffect );
ffeffect.dwFlags = FFEFF_CARTESIAN;
ffeffect.dwDuration = 3000000;
ffeffect.dwSamplePeriod = 1000;
ffeffect.dwGain = FF_FFNOMINALMAX;
ffeffect.dwTriggerButton = FFEB_NOTRIGGER;
ffeffect.dwTriggerRepeatInterval = 0;
ffeffect.cAxes = 1;
ffeffect.rgdwAxes = dwAxes;
ffeffect.rglDirection = lDirection;
ffeffect.lpEnvelope = &ffenvelope;
ffeffect.cbTypeSpecificParams = sizeof( ffperiodic );
ffeffect.lpvTypeSpecificParams = &ffperiodic;
ffeffect.dwStartDelay = 0;
```

Demo

TestFFAPI Tool

Fernando Urbina
nano@apple.com

Force Feedback Hardware Plug-In

- Abstracts the hardware
- Similar to FF drivers in other architectures
- It is a CFPlugIn (just like IOUSBLib and IOHIDLib)
- Use a code-less kext to match to your USB Interface
  - Merges a dictionary into the IOUSBInterface node
  - Dictionary is a UUID/Path key—value pair
  - Framework uses IOKit to search and load the plug-in

Code-Less kext Info.plist

Property List	Class	Value
CFBundleDevelopmentRegion	String	English
CFBundleGetInfoString	String	Version 1.0.0 Copyright _MyCompanyName_ 2002
CFBundleIconFile	String	
CFBundleIdentifier	String	com.MySoftwareCompany.driver.SampleForceFeedback
CFBundleInfoDictionaryVersion	String	6.0
CFBundleName	String	I/O Kit Driver for _MyCompanyName_ Force Feedback Devices
CFBundlePackageType	String	KEXT
CFBundleShortVersionString	String	1.0.0
CFBundleSignature	String	????
CFBundleVersion	String	1.0.0b1
▼IOKitPersonalities	Dictionary	1 key/value pairs
▼WizzyBang Force Feedback Device	Dictionary	8 key/value pairs
bConfigurationValue	Number	1
bInterfaceNumber	Number	0
CFBundleIdentifier	String	com.apple.iokit.IOUSBHIDDriver
idProduct	Number	65298
idVendor	Number	1699
▼IOCFPlugInTypes	Dictionary	1 key/value pairs
F4545CE5-BF5B-11D6-A4BI	String	SampleForceFeedback.kext/Contents/PlugIns/WizzyBang.plugin
IOClass	String	IOUSBHIDDriver
IOProviderClass	String	IOUSBInterface
▼OSBundleLibraries	Dictionary	2 key/value pairs
com.apple.iokit.IOHIDFamily	String	1.2.1
com.apple.iokit.IOUSBHIDDriver	String	1.9.4
OSBundleRequired	String	Safe Boot

Force Feedback Hardware Plug-In

- IOForceFeedbackLib.h defines the interface
- Generate your own UUID for your plug-in
  - Use the “uuidgen” tool in Mac OS X (man uuidgen)
- Interface file has good HeaderDoc
- Sample ForceFeedbackPlugIn project will be available

Summary

- Force Feedback Devices
- Force Feedback on Mac OS X Architecture
- ForceFeedback.framework
- Force Feedback Hardware Plug-in

Wrap Up

Roadmap

503 USB Update The latest from USB-land	**Marina** **Tue., 2:00pm**
506 CHUD Performance Optimization... Collect Data and Measure Performance	**Marina** **Wed., 2:00pm**
FF006 FireWire and USB Feedback Tell us what you think	**North Beach** **Wed., 3:30pm**

Who to Contact

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

Rob Yepez
I/O Kit Team
ryepez@apple.com

Fernando Urbina
USB Software Team
nano@apple.com

http://developer.apple.com/wwdc2003/urls.html

Q&A

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference